EMPIRICAL SUBSTRUCTURE DISCOVERY[1]

Lawrence B. Holder
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
405 North Mathews, Urbana, IL 61801

Abstract

This paper describes the substructure discovery method used in the SUBDUE system. The method involves a computationally constrained best-first search guided by four heuristics: cognitive savings, compactness, connectivity and coverage. The two main processes contained in this method are substructure generation and substructure selection. Substructure generation is the process by which new substructures are generated from previously considered substructures. The second process, substructure selection, chooses the best substructure among alternative substructures according to the four heuristics. Each of the four heuristics are described along with their role in the evaluation of a substructure. After the generation and selection processes are described, the substructure discovery algorithm is presented. Two examples demonstrate SUBDUE's ability to discover substructure and the advantages to be gained by other learning systems from the discovery of substructure concepts.

1 Introduction

The amount of detailed information available from a real-world environment is overwhelming. Yet, humans have the ability to ignore minute detail and extract information from the environment at a level of detail that is appropriate for the purpose of the observation [Witkin and Tenenbaum, 1983]. Machine learning systems that operate in such a detailed structural environment must be able to abstract over unnecessary detail in the input and determine which attributes are relevant to the learning task.

Substructure discovery is the process of identifying concepts describing interesting and repetitive "chunks" of structure within structural descriptions of the environment. Once discovered, the substructure concept can be used to simplify the descriptions by replacing all occurrences of the substructure with a single form that represents the newly discovered concept. The discovered substructure concepts allow abstraction over detailed structure in the original descriptions and provide new, relevant attributes for subsequent learning tasks.

This paper describes the substructure discovery method used in the SUBDUE system [Holder, 1988]. The SUBDUE system consists of a substructure discovery module, a substructure specialization module for specializing the substructures discovered by SUBDUE, and an incremental substructure background knowledge module that retains previously discovered substructures for use in subsequent learning tasks. Only the discovery module of SUBDUE is presented in this paper. Section 2 defines *substructure* and related terms. Section 3 discusses the substructure generation process, and Section 4 defines the heuristics used in the substructure selection process. Section 5 outlines SUBDUE's substructure discovery algorithm, and Section 6 illustrates some examples of SUBDUE's performance. Finally, Section 7 summarizes the substructure discovery process in SUBDUE and discusses future work.

2 Substructure

In a graphical sense, a substructure is a collection of nodes and edges comprising a connected subgraph of a larger graph. However, the substructures discovered by SUBDUE represent more than just a syntactic definition of a subgraph. Substructures are concepts. Substructure discovery is concerned with identifying

[1] A shorter version of this paper appears in the Proceedings of the Sixth International Workshop on Machine Learning (June, 1989).

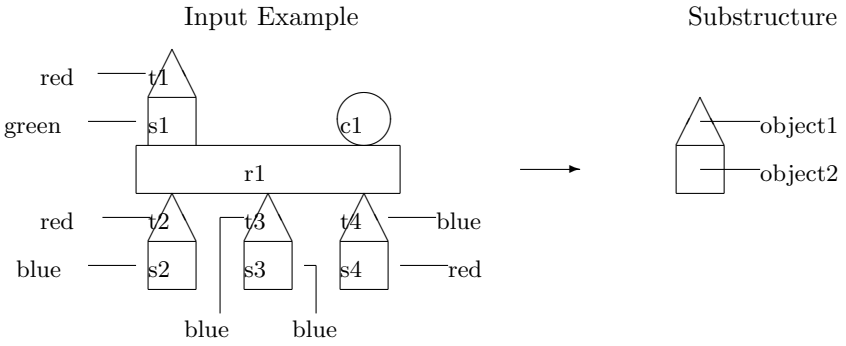


Figure 1: Example Substructure

substructures that represent interesting concepts, not just interesting graphical structure. Thus, substructures, or equivalently substructure concepts, should be interpreted as both collections of structurally related objects and as the conjunctive concepts describing them.

An appropriate language for describing substructures is an extension to the first order logic called Variable-valued Logic system 2 (VL_2) [Michalski, 1980], which is a subset of the Annotated Predicate Calculus (APC) [Michalski, 1983]. Figure 1 illustrates an input example along with the substructure discovered by SUBDUE. Both the input example and the substructure are expressed in the same substructure description language. The expression for the input example shown in Figure 1 is

<[shape(t1)=triangle][shape(t2)=triangle][shape(t3)=triangle][shape(t4)=triangle][shape(s1)=square]
[shape(s2)=square][shape(s3)=square][shape(s4)=square][shape(r1)=rectangle][shape(c1)=circle]
[color(t1)=red][color(t2)=red][color(t3)=blue][color(t4)=blue][color(s1)=green][color(s2)=blue]
[color(s3)=blue][color(s4)=red][on(t1,s1)=t][on(s1,r1)=t][on(c1,r1)=t][on(r1,t2)=t][on(r1,t3)=t]
[on(r1,t4)=t][on(t2,s2)=t][on(t3,s3)=t][on(t4,s4)=t]>

If each object of the substructure is assigned a symbolic name as in Figure 1 (e.g., object1, object2), then the expression for the substructure is

<[shape(object1)=triangle][shape(object2)=square][on(object1,object2)=t]>

A *substructure* is either a single object or a non-empty set of connected relations. The relations of a substructure are connected if the graph representation of the substructure, where objects are nodes and relations are edges in the graph, is connected. A *selector relation* consists of the selector relation name, a non-empty set of objects as arguments and the value of the selector relation. Selector relations are henceforth referred to as relations. An *object* is a primitive element from which relations and, ultimately, substructures are defined.

For the following discussions, some terminology is needed to describe important aspects of substructures as they relate to a given set of input examples. An *occurrence* of a substructure in a set of input examples is a set of objects and relations from the examples that match, graph theoretically, to the graphical representation of the substructure. For example, the occurrences of the substructure in the input example of Figure 1 are

<[on(t1,s1)=t][shape(t1)=triangle][shape(s1)=square]>
<[on(t2,s2)=t][shape(t2)=triangle][shape(s2)=square]>
<[on(t3,s3)=t][shape(t3)=triangle][shape(s3)=square]>
<[on(t4,s4)=t][shape(t4)=triangle][shape(s4)=square]>

A *neighboring relation* of an occurrence of a substructure is a relation in the input example that is not contained in the occurrence, but has at least one object from the occurrence as an argument. For example, the neighboring relations of the first occurrence listed above are [color(t1)=red], [color(s1)=green] and [on(s1,r1)=t].

An *external connection* of an occurrence of a substructure is a neighboring relation of the occurrence that has as an argument at least one object not contained in the occurrence. In other words, an external connection of an occurrence of a substructure is a relation that relates one or more objects in the occurrence to one or more objects not in the occurrence. For the first occurrence listed above, there is only one external connection, [on(s1,r1)=t].

3 Substructure Generation

An essential function of any substructure discovery system is the generation of alternative substructures. The substructure generation process constructs new substructures from the objects and relations in the input examples. SUBDUE's substructure discovery algorithm employs an approach to substructure generation called *minimal expansion*. An expansion approach begins with smaller substructures and expands them by appending additional structure from the input examples. Minimal expansion expands the substructures by appending the smallest amount of additional structure. In the context of substructures, this is equivalent to adding one neighboring relation. Thus, minimally expanding a substructure to form a new substructure involves appending one neighboring relation to the substructure. For example, according to the three neighboring relations of the occurrence, <[on(t1,s1)=t][shape(t1)=triangle][shape(s1)=square]>, the substructure in Figure 1 would be expanded to generate the following three substructures

<[shape(object1)=triangle][shape(object2)=square][on(object1,object2)=t][color(object1)=red]>
<[shape(object1)=triangle][shape(object2)=square][on(object1,object2)=t][color(object2)=green]>
<[shape(object1)=triangle][shape(object2)=square][on(object1,object2)=t][on(object2,object3)=t]>

SUBDUE uses an exhaustive minimal expansion technique for generating alternative substructures from a single substructure. The exhaustive version of this technique generates new substructures by considering all possible neighboring relations of the original substructure. To avoid the combinatorical explosion of this process, SUBDUE uses the substructure selection process to select the most promising substructure for expansion.

4 Substructure Selection

After using the method from the previous section to construct a set of alternative substructures, SUBDUE's substructure discovery algorithm chooses one of these substructures as the best hypothetical substructure. This is the task of substructure selection. The method of selection employs a heuristic evaluation function to order the set of alternative substructures based on their heuristic quality. This section presents the four heuristics used by SUBDUE to evaluate a substructure: cognitive savings, compactness, connectivity and coverage.

The first heuristic, *cognitive savings*, is the underlying idea behind several utility and data compression heuristics employed in machine learning [Wolff, 1982, Minton *et al.*, 1987, Whitehall, 1987] Cognitive savings measures the amount of data compression obtained by applying the substructure to the input examples. In other words, the cognitive savings of a substructure represents the net reduction in complexity after considering both the reduction in complexity of the input examples after replacing each occurrence of the substructure by a single conceptual entity and the gain in complexity associated with the conceptual definition of the new substructure. The reduction in complexity of the input examples can be computed as the number of occurrences of the substructure multiplied by the complexity of the substructure. Thus, the cognitive savings of a substructure, S, for a set of input examples, E, is computed as

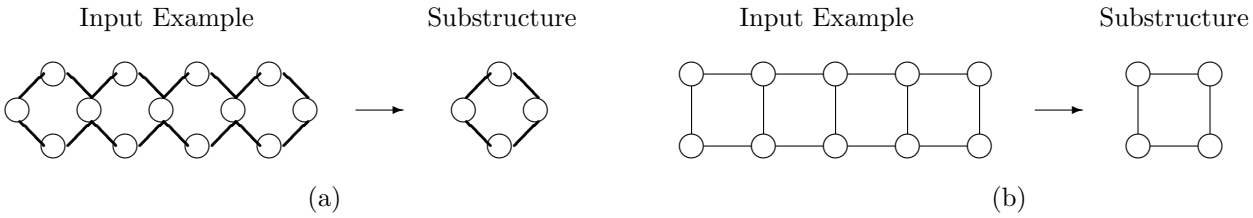


Figure 2: Overlapping Substructures

$$\begin{aligned} cognitive\_savings(S,E) &= complexity\_reduction(S,E) - complexity(S) \\ &= [\#occurrences(S,E) * complexity(S)] - complexity(S) \\ &= complexity(S) * [\#occurrences(S,E) - 1] \end{aligned}$$

In the above computation of cognitive savings the complexity of the substructure is typically a function of the number of objects, the number of relations, and the arity of the relations in the substructure. However, the number of occurrences of the substructure is more complicated to measure, because occurrences may overlap in the input examples. For instance, Figure 2 shows two input examples along with the substructure found by the discovery process. Here, the circles represent objects and the lines represent relations. At first glance, the number of occurrences of the substructure in Figure 2a may appear to be four; however, the number of non-overlapping occurrences is less than four. Figure 2a illustrates the problem of object overlap, and Figure 2b illustrates the problem of relation overlap. In view of the overlap problem, computation of the number of occurrences must reflect the number of unique occurrences.

In SUBDUE's substructure discovery algorithm, the $complexity(S)$ is defined as the size of the substructure, S, where the size is computed as the sum of the number of objects and relations in the substructure. As discussed above, the $\#occurrences(S,E)$ is more complicated to compute, because the occurrences may overlap in the input examples. In view of the overlap problem, simply counting all objects and relations in the overlapping occurrences would incorrectly state the true cognitive savings of the substructure. Therefore, the $complexity\_reduction(S,E)$ is redefined to be the number of objects and relations in the occurrences of the substructure, where overlapping objects and relations are counted only once. The number of such objects is referred to as $\#unique\_objects$, and the number of such relations is referred to as $\#unique\_relations$. Thus, the cognitive savings of a substructure, S, with occurrences, OCC, in the set of input examples, E, is computed as

$$\begin{aligned} cognitive\_savings(S,E) &= complexity\_reduction(S,E) - complexity(S) \\ &= [\#unique\_objects(OCC) + \#unique\_relations(OCC)] - complexity(S) \\ &= [\#unique\_objects(OCC) + \#unique\_relations(OCC)] - size(S) \\ &= [\#unique\_objects(OCC) + \#unique\_relations(OCC)] - [\#objects(S) + \#relations(S)] \end{aligned}$$

As an example of the cognitive savings calculation, consider the input examples and corresponding substructures in Figure 2. If each circle is considered an object and each line a relation, then for each of the two substructures, $\#objects(S) = 4$, $\#relations(S) = 4$, and there are four occurrences of the substructure in the input example. In Figure 2a, $\#unique\_objects(OCC) = 13$ and $\#unique\_relations(OCC) = 16$; thus, $cognitive\_savings = [13 + 16] - [4 + 4] = 21$. In Figure 2b, $\#unique\_objects(OCC) = 10$ and $\#unique\_relations(OCC) = 13$; thus, $cognitive\_savings = [10 + 13] - [4 + 4] = 15$.

The second heuristic, *compactness*, measures the "density" of a substructure. This is not density in the physical sense, but the density based on the number of relations per number of objects in a substructure. The compactness heuristic is a generalization of Wertheimer's *Factor of Closure*, which states that human attention is drawn to closed structures [Wertheimer, 1939]. Graphically, a closed substructure has at least as many relations as objects, whereas a non-closed substructure has fewer relations than objects [Prather, 1976].

Thus, closed substructures have a higher compactness value. Compactness is defined as the ratio of the number of relations in the substructure to the number of objects in the substructure.

$$compactness(S) = \frac{\#relations(S)}{\#objects(S)}$$

For each of the substructures in Figure 2, $\#relations(S) = 4$ and $\#objects(S) = 4$; thus, $compactness = 4/4 = 1$.

The third heuristic, *connectivity*, measures the amount of external connection in the occurrences of the substructure. The connectivity heuristic is a variant of Wertheimer's *Factor of Proximity* [Wertheimer, 1939], and is related to earlier numerical clustering techniques [Zahn, 1971]. These works demonstrate the human preference for "isolated" substructures, that is, substructures that are minimally related to adjoining structure. Connectivity measures the "isolation" of a substructure by computing the average number of external connections over all the occurrences of the substructure in the input examples. The number of external connections is to be minimized; therefore, the connectivity value is computed as the inverse of the average to arrive at a value that increases as the number of external connections decreases. Thus, the connectivity of a substructure, S, with occurrences, OCC, in the set of input examples, E, is computed as

$$connectivity(S, E) = \left[\frac{\sum_{i \in OCC} |external\_connections(i)|}{|OCC|}\right]^{-1}$$

Again consider Figure 2. Each substructure has four occurrences in the input example. For both substructures the two innermost occurrences both have 4 external connections and the two outermost occurrences both have 2 external connections, for a total of 12 external connections. Thus, $connectivity = (12/4)^{-1} = 1/3$.

The final heuristic, *coverage*, measures the amount of structure in the input examples described by the substructure. The coverage heuristic is motivated from research in inductive learning and provides that concept descriptions describing more input examples are considered better [Michalski and Stepp, 1983]. Coverage is defined as the number of unique objects and relations in the occurrences of the substructure divided by the total number of objects and relations in the input examples. Thus, the coverage of a substructure, S, with occurrences, OCC, in the set of input examples, E, is computed as

$$coverage(S, E) = \frac{\#unique\_objects(OCC) + \#unique\_relations(OCC)}{\#objects(E) + \#relations(E)}$$

For both substructures in Figure 2 the occurrences of the substructure describe every object and relation in the input example; thus, $coverage = 1$.

Ultimately, the value of a substructure, S, for a set of input examples, E, is computed as the product of the four heuristics.

$$value(S, E) = cognitive\_savings(S, E) * compactness(S) * connectivity(S, E) * coverage(S, E)$$

In this way the compactness, connectivity and coverage heuristics refine the cognitive savings by increasing or decreasing the total value to reflect specific qualities of the substructure. Thus, for the substructure in Figure 2a, $value = 21 * 1 * 1/3 * 1 = 7.0$; and for the substructure in Figure 2b, $value = 15 * 1 * 1/3 * 1 = 5.0$. Applying the heuristic evaluation to the substructure of Figure 1, $value = 15 * 3/2 * 1/3 * 20/37 = 4.054$.

5 Substructure Discovery Algorithm

Ideally, an algorithm for discovering substructure should converge on the best substructure in terms of the goal of the discovery task. The goal of the substructure discovery algorithm, in general, is to identify the substructure in the input examples that maximizes the capacity for complexity reduction and maximizes the interestingness of the substructure concept. SUBDUE measures both these characteristics with the heuristic evaluation function defined in Section 4. However, the number of possible substructures is exponential in the number of relations within the given input examples. If left unconstrained, the algorithm may eventually

consider all possible substructures. SUBDUE imposes a computational limit on the algorithm to constrain the number of substructures considered.

The substructure discovery algorithm used by SUBDUE is a computationally constrained best-first search guided by the substructure generation and selection processes. The algorithm is given one or more input examples and a limit on the amount of computation performed. The algorithm begins by forming the set, S, of alternative substructures. Initially, the set has only one element, the substructure corresponding to a single object, with as many occurrences as there are objects in the input examples. As the algorithm progresses, the discovered substructures are kept in the set, D, which is initially empty.

The next step in the algorithm is a loop that continuously generates new substructures from the substructures in S until either the computational limit is exceeded or the set of alternative substructures, S, is exhausted. The loop begins by selecting the best substructure in S. Here, the value computation of Section 4 is employed to choose the best substructure from the alternatives in S. Once selected, the best substructure is stored in $BESTSUB$ and removed from S. Next, if $BESTSUB$ does not already reside in the set D of discovered substructures, then $BESTSUB$ is added to D. The substructure generation method of Section 3 is then used to construct a set of new substructures by minimally expanding $BESTSUB$. The newly generated substructures that have not already been considered by the algorithm are added to S, and the loop repeats. When the loop terminates, D contains the set of discovered substructures.

Thus, the substructure discovery algorithm searches for the heuristically best substructure until all possible substructures have been considered or the amount of computation exceeds the given limit. Due to the large number of possible substructures, the algorithm typically exhausts the allotted computation before considering all possible substructures. Therefore, the algorithm may not find the substructure that maximizes the heuristic evaluation function. However, experiments in a variety of domains indicate that the heuristics perform well in guiding the search toward more promising substructures [Holder, 1988].

6 Examples

This section presents two examples that demonstrate SUBDUE's ability to discover substructure and the advantages to be gained by other learning systems from the discovery of substructure concepts. Each example is run on a Texas Instruments Explorer using a Common Lisp implementation of the SUBDUE system.

6.1 Example 1

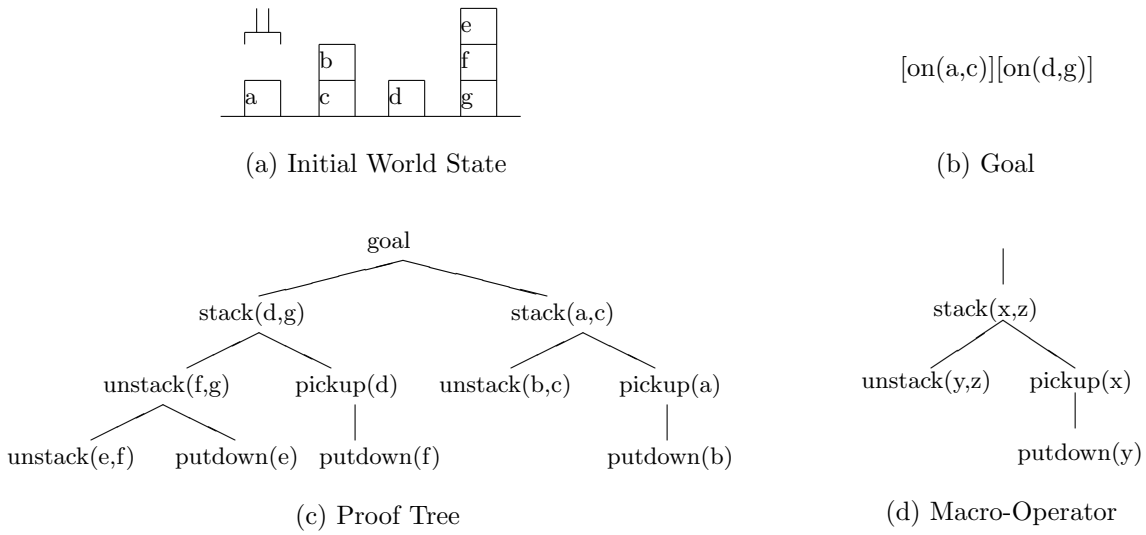
Example 1 illustrates a possible application of the substructure discovery algorithm to the task of discovering macro-operators in plans. The example is drawn from the "blocks world" domain. The operators for this domain are taken from [Nilsson, 1980]: **pickup**, **putdown**, **stack** and **unstack**.

For this example, suppose the initial world state is as shown in Figure 3a, and the desired goal is in Figure 3b. The proof tree of operators to achieve the goal is shown in Figure 3c. With this proof tree as input, SUBDUE discovers the substructure shown in Figure 3d after considering 19 alternative substructures. The substructure represents a macro-operator for accomplishing a subgoal to stack a block, x, on another block, z, when a block, y, is already on top of block z.

Macro-operators discovered by SUBDUE can be used in several ways. Replacing the occurrences of the macro-operator in the original proof tree by instantiations of the macro-operator can reduce the storage requirements of the schema constructed from the entire proof tree. Retaining the macro-operators discovered within a proof tree would provide sub-schemas in addition to the schemas learned by an explanation-based learning (EBL) system [Mitchell *et al.*, 1986, DeJong and Mooney, 1986]. The sub-schemas would increase the amount of operationalized knowledge available to the EBL system for explaining subsequent examples.

6.2 Example 2

Example 2 combines SUBDUE with the INDUCE system [Hoff *et al.*, 1983] to demonstrate the improvement gained in both processing time and quality of results when the examples contain a large amount of structure.



(a) Initial World State

(b) Goal

(c) Proof Tree

(d) Macro-Operator

Figure 3: Proof Tree Example

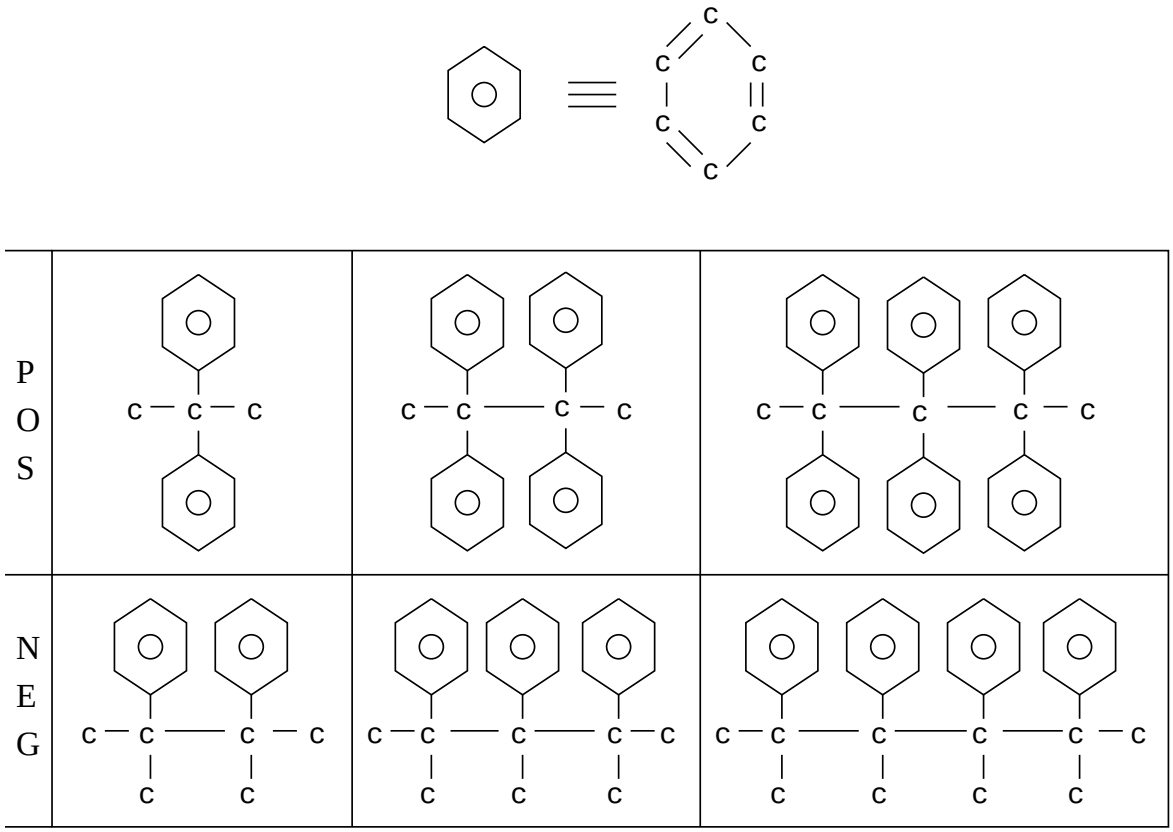
A Common Lisp version of INDUCE was used for this example running on the same Texas Instruments Explorer as the SUBDUE system.

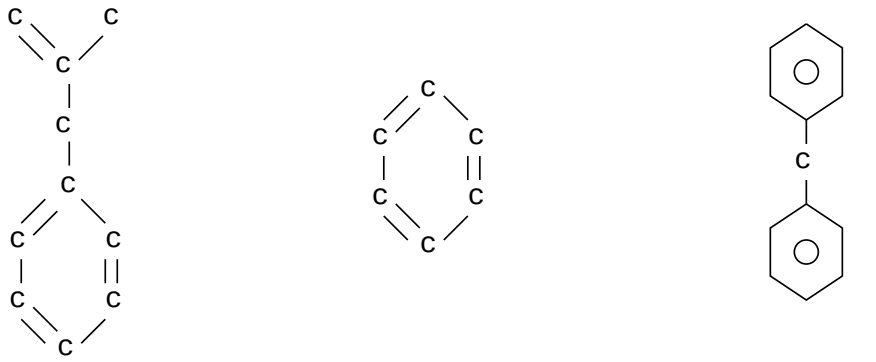
Figure 4a shows a pictorial representation of the three positive and three negative examples given to INDUCE. Each of the symbolic benzene rings in Figure 4a represents the more complex structure in the left side of Figure 4c. The actual input specification for the six examples contains a total of 178 relations of the form [single-bond(c1,c2)=t] or [double-bond(c1,c2)=t]. After 160 seconds of processing time, INDUCE produces the concept shown in Figure 4b. Next, all six examples are given to SUBDUE using the same 178 relations. After considering seven alternative substructures for 15 seconds of processing time, SUBDUE discovers the substructure concept of a benzene ring as shown on the left side of Figure 4c. The newly discovered substructure is then used to reduce the complexity of the original examples by replacing each occurrence of the benzene ring with a single relation, i.e., [benzene-ring(c1,c2,c3,c4,c5,c6)=t]. Using the reduced set of positive and negative examples, INDUCE produces the concept on the right side of Figure 4c in 38 seconds of processing time. Here, the symbolic benzene rings represent the *benzene-ring* relation, not the complex structural representation used in the original descriptions of the examples.

By abstracting over the structure representing the benzene ring, SUBDUE allows INDUCE to discover the true concept distinguishing the positive and negative examples; namely, benzene rings are paired across one carbon atom in the positive examples, but not in the negative examples. INDUCE represents this concept in terms of the abstract benzene ring feature provided by SUBDUE. Furthermore, the processing time of SUBDUE and INDUCE combined (53 seconds) represents a speedup of 3.0 over that of INDUCE alone. This example demonstrates how the substructures discovered by SUBDUE can improve the results of other learning systems by abstracting over detailed structure in the input and providing new features.

7 Conclusion

This paper describes the method used by the SUBDUE system to discover substructures in structured examples. The method involves a computationally constrained best-first search guided by four heuristics: cognitive savings, compactness, connectivity and coverage. Alternative substructures are generated by the minimal expansion technique that constructs new substructures by adding minimal structure to previously considered substructures. The two examples demonstrate SUBDUE's ability to find plausible substructures





160 sec.

INDUCE

15 sec.

SUBDUE

38 sec.

INDUCE

Figure 4: SUBDUE/INDUCE Example

and the possible uses of these substructures by other learning systems.

Earlier work in substructure discovery can be found in Winston's ARCH program [Winston, 1975]. Winston used several domain dependent methods to identify recurring structure in the blocks world examples. Recent work in substructure discovery includes Whitehall's PLAND system for discovering substructure in action sequences [Whitehall, 1987]. Whitehall uses the cognitive savings heuristic along with three levels of background knowledge to discover loops and conditionals in the sequences.

In addition to the substructure discovery module, SUBDUE also contains a substructure specialization module and a substructure background knowledge module. Substructures discovered by SUBDUE are specialized by adding additional structure. Both the original and specialized substructures are stored hierarchically in the background knowledge. The background knowledge may then direct the discovery process towards substructures similar to those already known. Future work and experimentation is necessary to evaluate the improvements gained by using the specialization and background knowledge modules and to incorporate other forms of background knowledge into SUBDUE's substructure discovery process.

Acknowledgements

The author would like to thank Robert Stepp for his helpful suggestions throughout this work. This research was partially supported by the Defense Advanced Research Projects Agency under grant N00014-87-K-0874 and by a gift from Texas Instruments, Inc.

References

[DeJong and Mooney, 1986] G. F. DeJong and R. J. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, April 1986.

[Hoff *et al.*, 1983] W. A. Hoff, R. S. Michalski, and R. E. Stepp. Induce 3: A program for learning structural descriptions from examples. Technical Report UIUCDCS-F-83-904, Department of Computer Science, University of Illinois, 1983.

[Holder, 1988] L. B. Holder. Discovering substructure in examples. Master's thesis, Department of Computer Science, University of Illinois, Urbana, IL, May 1988.

[Michalski and Stepp, 1983] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 331–363. Tioga Publishing Company, 1983.

[Michalski, 1980] R. S. Michalski. Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4):349–362, 1980.

[Michalski, 1983] R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Tioga Publishing Company, 1983.

[Minton *et al.*, 1987] S. Minton, J. G. Carbonell, O. Etzioni, C. A. Knoblock, and D. R. Kuokka. Acquiring effective search control rules: Explanation-based learning in the PRODIGY system. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 122–133, 1987.

[Mitchell *et al.*, 1986] T. M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, January 1986.

[Nilsson, 1980] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[Prather, 1976] R. Prather. *Discrete Mathemetical Structures for Computer Science*. Houghton Miffin Company, 1976.

[Wertheimer, 1939] M. Wertheimer. Laws of organization in perceptual forms. In W. D. Ellis, editor, *A Sourcebook of Gestalt Psychology*, pages 331–363. Harcourt, Brace and Company, 1939.

[Whitehall, 1987] B. L. Whitehall. Substructure discovery in executed action sequences. Master's thesis, Department of Computer Science, University of Illinois, 1987.

[Winston, 1975] P. H. Winston. Learning structural descriptions from examples. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 157–210. McGraw-Hill, 1975.

[Witkin and Tenenbaum, 1983] A. P. Witkin and J. P. Tenenbaum. On the role of structure in vision. In J. Beck, B. Hope, and A. Rosenfeld, editors, *Human and Machine Vision*, pages 481–543. Academic Press, 1983.

[Wolff, 1982] J. G. Wolff. Language acquisition, data compression and generalization. *Language and Communication*, 2(1):57–89, 1982.

[Zahn, 1971] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, 20(1):68–86, 1971.